QU at TREC-2015: Building Real-Time Systems for Tweet Filtering and Question Answering

Reem Suwaileh, Maram Hasanain, Marwan Torki, Tamer Elsayed

Computer Science and Engineering Department
Qatar University
Doha, Qatar
{reem.suwaileh,maram.hasanain,mtorki,telsayed}@qu.edu.qa

ABSTRACT

This paper presents our participation in the microblog and LiveQA tracks in TREC-2015. Both tracks required building a "real-time" system that monitors a data stream and responds to users' information needs in real-time.

For the microblog track, given a set of users' interest profiles, we developed two online filtering systems that recommend "relevant" and "novel" tweets from a tweet stream for each profile. Both systems simulate real scenarios: filtered tweets are sent as push notifications on a mobile phone or as a periodic email digest. We study the effect of using a static versus dynamic relevance thresholds to control the relevancy of filtered output to interest profiles. We also experiment with different profile expansion strategies that account for potential topic drift. Our results show that the baseline run of the push notifications scenario that uses a static threshold with light profile expansion achieved the best results. Similarly, in the email digest scenario, the baseline run that used a shorter representation of the interest profiles without any expansion was the best run.

For the LiveQA track, the system was required to answer a stream of around 1000 real-time questions from Yahoo! Answers. We adopted a very simple approach that searched an archived Yahoo! Answers QA dataset for similar questions to the asked ones and retrieved back their answers.

1. INTRODUCTION

Twitter has rapidly developed over the past years and become a massive information sharing network. It gained a reputation for carrying the *heartbeat* of the world by allowing users to continuously post and read tweets about current events and news. With a huge flood of tweets shared daily, users are overwhelmed by the amount of data they need to follow in order to track a certain event or a topic. Automatically-satisfying the user interest in following a certain topic over a non-stop stream of tweets is challenging as it requires a real-time system that *filters* relevant and novel tweets from a rapidly flowing stream. Moreover, the filtering system is required to address problems originating from the nature of tweets such as their limited number of characters; with maximum 140 characters per tweet, and their conversational and temporal characteristics. The shortness of tweets causes a challenge known as sparsity and the change of the topic over time stems another dominant challenge called drifting challenge.

There have been several studies for applying online filtering on Twitter stream that attempts to overcome these challenges e.g [1, 3]. The first study adapted the state-of-the-art news filtering technique that uses Incremental Rocchio classifier to continuously update the tracked topics on the tweets stream [1]. The proposed adaptation encounter the sparsity problem by expanding the userâĂŹs profile with relevant and recent terms to enrich the interest representation. However, along with this solution, the possibility of topic drift arises especially that the tweets' stream is hastily developing and the topic representation should keep-up with that. Thus, in addition to investigating an event detection approach to detect the topic drift in the tweet stream, an attempt to balance the contribution of the long-term and short-term interest to the interest profile was applied. The adaptation was evaluated using TREC Microblog track 2012 and it showed the effectiveness of this adaptation, but not to the extend achieved in news filtering. Another study experimented different smoothing models to integrate a background and a foreground language models (LM) that represent the tracked topics in the stream. Both LMs were trained per-topic using hashtags from Twitter stream [3]. The background language model represents the topic tracking on the continuous stream for about one month where the foreground language model represents the recently captured changes on the topic from the same stream. The approach was examined over ten topics and approved that it performs well in the filtering task.

In this work, we propose tweet filtering systems that attempts to handle these challenges for both scenarios in the TREC-2015 microblog track: push notifications on a mobile phone and a periodic email digest. The problem, our approach and evaluation results are further discussed in Section 2.

As with Twitter, social question answering (sQA) systems are increasingly gaining importance and attracting millions of users to post and answer questions. Yahoo! Answers[1] is by far one of the largest sQA platforms. Questions and answers on such platforms share some characteristics with tweets in terms of being conversational and often very socially-oriented.

In the past few years, QA as a research problem has received significant attention, however, building effective au-

[1]https://answers.yahoo.com/

tomatic QA is impeded by a lack of suitable datasets to train these systems. Automatic QA is even more challenging when the focus goes beyond seeking short answers to factoid questions, to questions that require reasoning, explanations, etc. The datasets used in such previous systems are either too small to train a good system or not extensive to involve multiple domains [5]. However, the existence of large social question answering websites, such as Yahoo! Answers specifically, makes the development of automated answering systems more interesting due to the scale of the available datasets. But the amount of questions posted on such platforms is not usually matched by the number of answers provided; many questions are left unanswered and many are repeated. Automatically answering questions in such cases is a useful feature users of these sQA can benefit from which intrigues a need for designing automatic sQA systems. This is in fact the problem we tackle as part of the LiveQA track in TREC-2015. We discuss our approach and evaluation results for the LiveQA track in Section 3.

2. REAL-TIME TWEET FILTERING

Tweet filtering is the reverse of ad-hoc tweet search, where a user provides an information need at a certain point in time, and the filtering system is expected to filter relevant tweets to the information need from a stream of tweets posted after the query time.

Tweet filtering was first introduced in the microblog track in TREC-2012 [4]. The task ran on a simulated stream of tweets from a collection of 16M tweets [4] that was crawled prior to the running of the task. Differently, the microblog track in TREC-2015 ran a real-time tweet filtering task that worked on a real stream of tweets in a 10-day evaluation period. Moreover, the 2015 task has two modes: 1) a scenario that pushes few filtered tweets as push notifications on a mobile phone to the user, and 2) a scenario where a periodic email digest of a 10-times larger set of top relevant tweets to the topic is sent to user [2].

In both scenarios, the filtering system is expected to send a set of *novel* (i.e., non-redundant) and *relevant* (i.e., on-topic) tweets every day.

In this task, an *interest profile* that is composed of a title, description, and narrative is used as a representation of the user information need (or topic). The title is short, having few keywords to represent the topic, the description is a one sentence statement of the information need of the user, and the narrative is a full paragraph describing that information need.

2.1 Approach

In this section, we first present the main approach of our solution for both scenarios, and then discuss further details that are specific to each of them. Figure 1 shows a high-level overview of the core filtering system on which we build the systems for the two scenarios.

2.1.1 Core System Design

Cold Start and Preprocessing: Following the track guidelines, we only consider English tweets for filtering. The filtering system filters English tweets using an open-source language detection tool[2]. After that, it performs simple preprocessing on tweets and profiles including stemming, stopwords

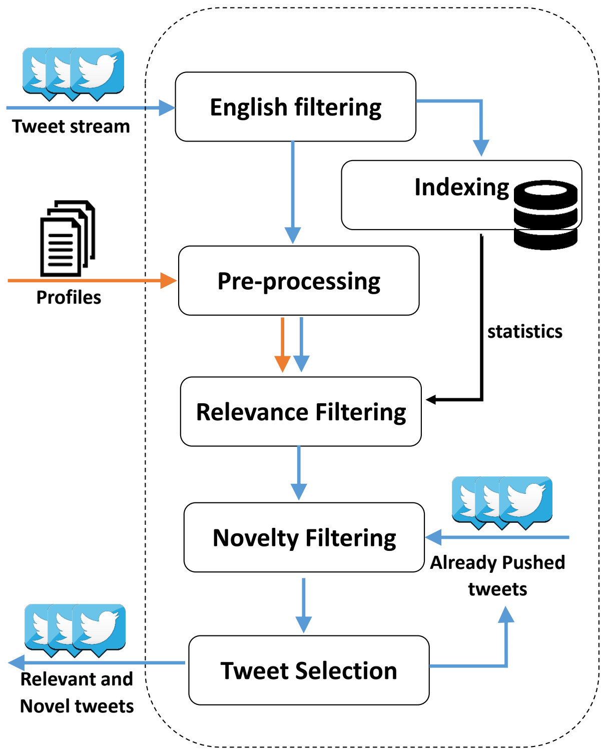

Figure 1: An Overview of the core real-time tweet filtering system

removal, and URL removal. The system represent tweets and profiles as vectors of terms weighted using inverse document frequency (idf)-based term weighting. Term weights are computed using the equation below:

$$idf(term) = \log \frac{N - df(term) + 0.75}{df(term) + 0.75} \tag{1}$$

Where N is the total number of tweets in the collection of tweets from which the system is extracting term statistics and $df(term)$ is the document frequency of the term. As the equation shows, term weights are computed using a history of past tweets which is naturally not available when the system first starts. To solve this “cold start” problem, the system is initialized with an index[3] of a 3-day stream of tweets preceding the beginning of the evaluation period. Additionally, the system periodically indexes the incoming tweets during the evaluation period.

Relevance and Novelty: For both scenarios, we used the following approach for filtering based on tweet relevance and novelty. Given an interest profile for a topic, the filtering system processes the incoming stream one tweet at a time. For each tweet, a relevance score to each interest profile is computed using Cosine similarity. A tweet with a similarity score above a relevance threshold τ_r is considered relevant to the topic. However, the relevant tweet is not pushed to the user unless its novelty score is less than a novelty threshold τ_n. Given a list of filtered tweets that were sent to user over

[2]https://code.google.com/p/language-detection/

[3]The index is created and updated using Apache Lucene 4.5.0 (https://lucene.apache.org/)

previous days, the system computes the similarity between the incoming tweet and each of the tweets in that list using a modified version of Jaccard similarity. If the similarity of the tweet to any of the tweets in the list exceeds the novelty threshold τ_n, then the system considers the tweet as redundant and elects not to send it to user. Otherwise, the tweet is considered both relevant and novel, and thus the system flags it to be sent to the user.

Profile Expansion: Since topics are tracked over a period of time, the system periodically updates the topic representation to reflect the topic development over that period. To achieve that, the system performs profile expansion using pseudo relevance feedback; expansion terms are extracted from pseudo relevant tweets filtered before the expansion time. The expanded profile vector is computed using the following equation:

$$\vec{q}^{'} = \vec{q} + \beta(\vec{e}) \tag{2}$$

Where $\vec{q}^{'}$ is the expanded profile vector, $\vec{q}$ is the initial profile vector created when the system first starts and $\vec{e}$ is the vector of expansion terms extracted from the pseudo relevant tweets for each profile independently. Common terms between the extracted expansion terms and the profile title terms are eliminated before creating $\vec{e}$. Parameter β is used to control the contribution of expansion terms to the final profile vector.

2.1.2 Scenario A: Push Notifications

This scenario simulates a situation where updates about a topic are sent as mobile notifications to the user. The task requires pushing a maximum of 10 tweets per day not to overload the user with notifications. The main challenge in pushing such small number of tweets is that the system has very few opportunities to push high quality tweets to the user. To ensure pushing high quality tweets, the system should consider how fresh the tweets are in addition to how relevant and novel they are. In this context, the freshness of tweets is measured as the elapsed time between the tweet's creation time (the time when tweet appears in the stream) and the time it was pushed to the user.

Profile Representation: We represent the initial interest profile as a vector of terms $\vec{q}$ modeled as follows:

$$\vec{q} = \vec{title} + \alpha(\vec{desc} + \vec{narr}) \tag{3}$$

Where $\vec{title}$, $\vec{desc}$ and $\vec{narr}$ are the vectors of the title, description, and narrative of the profile respectively. The title is given higher weight since it contains the most concise and relevant description of the topic while the description and narrative contribution to the topic is controlled by the parameter α (< 1). Since narrative and description are much longer than the title and might share terms, we further control their contribution to the topic vector by selecting top k weighted terms of the resulting vector $(\vec{desc} + \vec{narr})$ to be added to the final topic vector.

Selecting Tweets to Push: Following all interest profiles in parallel, the system monitors the tweet stream and filters tweets based on their similarity and novelty. Over time, the system maintains a list of relevant and novel tweets for each profile and only pushes a tweet from that list periodically with a time period of length δ or when the size of the list exceeds a limit l.

After scoring all tweets in that list using equation 4 below, the system pushes the top-scoring tweet to the user. The computed tweet score considers how relevant it is to the profile and how fresh as well.

$$S(t) = S_r(t) * \frac{100 - (CurTime - time(t))}{100} \tag{4}$$

$S_r(t)$ is the relevance score of tweet t computed using cosine similarity between a profile and the tweet, $CurTime$ is the current system time, and $time(t)$ is the tweet creation time.

Threshold Updates: We experiment with using both static and dynamic relevance threshold settings. In the static threshold mode, the relevance threshold τ_r is set before the system starts and it does not change during the evaluation period. As for the dynamic threshold mode, the system starts with an initial threshold for each interest profile p_i and updates per-profile relevance threshold τ_r periodically. To update the threshold, the system maintains the number of tweets found relevant per profile in the last period. If the profile p_i gets no relevant tweets in the past time period, then the relevance threshold τ_r is decreased by 0.025 with a lower bound of 0.5. Otherwise, the threshold is increased using the following equation:

$$\tau_{r_i}^{'} = \tau_{r_i} + min(\frac{R_{p_i}}{100}, 0.15) \tag{5}$$

Where τ_{r_i} is the current threshold of profile p_i, $\tau_{r_i}^{'}$ is the updated threshold of that profile and R_{p_i} is the number of relevant tweets filtered for profile p_i within a time period t_T. The threshold upper bound is set to 0.95.

2.1.3 Scenario B: Periodic E-mail Digest

This scenario simulates a filtering system that daily sends a list of m relevant and non-redundant tweets from the tweet stream to the user given an interest profile. Initial profiles are represented in the vector space by the weighted title terms:

$$\vec{q} = \vec{title} \tag{6}$$

Composing E-mail Digest: After the end of each day in the evaluation period, the system issues all profiles as queries against an *Apache Lucene* search engine that searches the system's tweets index. The search engine uses query-likelihood with Dirichlet smoothing to retrieve a ranked list of the most relevant $2m$ tweets for each profile. Given this list for each query, the system extracts top m novel tweets and sends them to the user as an email digest. The novelty score of a tweet is computed by its similarity to all tweets previously sent to the user. If the similarity exceeded novelty threshold τ_n, the tweet is treated as redundant and then discarded, otherwise the tweets is added to the email digest to be sent.

Profile Expansion: The main focus in this scenario is to explore different sources of expansion terms for a profile. The baseline run does not perform expansion, but the remaining runs use different sources of expansion. In addition to extracting the top pseudo relevant tweets returned by searching the index as equation 2, the other expansion approach also uses the top weighted terms from description and narrative fields of the interest profile. In both approaches of expansion, the pseudo relevant tweets are extracted after filtering out tweets that exceeds the novelty threshold τ_n, the terms of the novel tweets are ranked and the top t weighted ones are added to the profile vector as follows:

$$\vec{q'} = \vec{q} + \alpha(\vec{desc} + \vec{narr}) + \beta\vec{e} \quad (7)$$

Where the $\vec{desc}$ and $\vec{narr}$ are the terms vector of the description and narrative fields of the interest profile respectively and their contribution to the profile is controlled by α parameter. Additionally, any common terms between the extracted terms and the initial profile terms are eliminated.

2.2 Experimental Evaluation

2.2.1 Evaluation Measures

The push notification scenario is evaluated using two main measures: the primary measure which is expected latency-discounted gain (ELG) and the normalized cumulative gain (nCG) [2]. Measures can be computed as follows:

$$ELG = \frac{1}{R_t} \sum Gain(t_i) \quad (8)$$

$$nCG = \frac{1}{Z} \sum Gain(t_i) \quad (9)$$

where $Gain(t_i)$ is 0 if the tweet is not relevant, a spam or junk, or 0.5 if the tweet is somewhat interesting, otherwise, it is set to 1. The gain of a tweet is reduced using the following time latency penalty:

$$latency = max(0, \frac{100 - delay}{100}) \quad (10)$$

where delay is the difference in minutes between tweet creation time and tweet push time.

As for the e-mail digest scenario, the e-mail digest is evaluated as a ranked-list of tweets. The evaluation measure used in this task is the normalized discounted cumulative gain (NDCG) computed at length k of this list.

For all evaluation measures in all scenarios, the score of a topic is the average of the measure values over all days and a score of a run is the average of that across all topics.

2.2.2 Submitted Runs

In each day of the evaluation period, the filtering system of scenario A pushes a maximum of $n = 10$ tweets to the user. Where the in the email digest scenario, the system sends an email digest with a list of $m = 100$ tweets. This section describes our submitted runs for both scenarios. Table 2.2.2 shows the different configuration settings is each run.

Scenario A:

- **QUBaseline:** is a baseline that uses a static relevance threshold τ_r when comparing a tweet to a profile. The system in this run represents the interest profile by title, narrative and description. The parameter $\alpha = 0.2$ controls the influence of the terms extracted from both description and narrative fields in profile representation, and 8 terms from these two fields are finally added to the profile. The profile is periodically expanded using a maximum of 4 expansion terms extracted from pseudo relevant tweets with $\beta = 0.2$.

- **QUDyn:** this run has a similar configuration to QUBaseline, except that the relevance threshold is updated dynamically for each profile.

- **QUDynExp:** this run is similar to QUDynm, but the number of expansion terms extracted from pseudo relevant tweets is set to 12. The system selects 10 expansion terms from narrative and description fields to include in profile representation and uses $\beta = \alpha = 0.3$.

Table 2 shows the results for the push notification scenario. The score of a run is computed as an average score of all profiles over all days. It can be clearly seen that the baseline run outperforms the other runs.

Table 2: Results for runs of the tweet push notification scenario

Run	ELG	nCG
QUBaseline	0.2750	0.2347
QUDyn	0.1850	0.1762
QUDynExp	0.1848	0.1763

Scenario B:

- **QUBaselineB:** is the baseline run where the system filters tweets using the initial profiles representation shown in equation 6.

- **QUExpB:** in this run, the system extracts the expansion terms from the top pseudo relevant tweets returned by searching the index where these terms do not overlap with the profile's title terms.

- **QUFullExpB:** this run is similar to QUExpB but it also expands the profile using top weighted terms obtained from the description and narrative fields of the profile. The number of top terms taken from both sources is controlled by $\alpha = 0.3$ and $\beta = 0.2$ parameters in order to balance their influence. Additionally, the maximum number of terms in the expanded profile is set to be ≥ 20 terms.

The performance of all runs is somewhat similar, however, the baseline run has a slightly better performance. We further investigated the poor performance of runs in scenario B and discovered a bug in our system that we think is behind the bad results. We fixed this issue in the system post-TREC submission and re-ran the system. Table 3 shows the NDCG results of submitted runs and the post-submission results for the runs of this scenario.

Table 3: NDCG results of the e-mail digest scenario

Run	submitted runs	post-TREC
QUBaselineB	0.1288	0.2251
QUExpB	0.1180	0.1954
QUFullExpB	0.1196	0.1811

3. LIVEQA

The LiveQA task is introduced this year for the first time in TREC. In this track, systems are supposed to return answers to real-time questions originating from real users via a live question stream. The language for the track is English for both questions and answers.

Several restrictions are applied to the track to make it more challenging. First restriction is the maximum allowed

Table 1: Summary of submitted runs for the two scenarios

Scenario	Run	Dynamic τ_r?	Base τ_r	Expansion?	# Pseudo tweets	# Expansion terms
Mobile Push Notification	QUBaseline	✗	0.6	✓	20	4
	QUDyn	✓	0.8	✓	20	4
	QUDynExp	✓	0.8	✓	20	12
Periodic Email Digest	QUBaselineB	✗	-	✗	-	-
	QUExpB	✗	-	✓	5	3
	QUFullExpB	✗	-	✓	10	10

response time, which is set to be one minute only. Second is the maximum answer length, which is set to just 1000 characters. Third is the limit on returned answers by the system, which is set to be the top retrieved ranked answer by the system.

The approach we followed for this task was to implement a simple system, which can be considered as a baseline for our future work on that problem/track for next year(s). The intuition behind that approach is also simple; since the expected questions are as of same type asked on Yahoo! Answers, we chose to use only Yahoo! Answers as the source of retrieving answers.

3.1 System Pipeline

We describe the pipeline for question answering as follows:

1. **Build/get access to an archive of questions and corresponding answers.** In this step, we use the question answers dataset provided by [5].

2. **Index the archived the question answer dataset.** In this step we use Lucene 4.7.0[4] with stop words removal. The objective here is to search the questions answers dataset for similar questions in real time.

3. **Retrieve potential answers.** For a newly-asked question, we search the pre-built index to find similar question(s). Answers for similar questions are retrieved and considered as potential answers for the asked question.

4. **Limiting the answer size.** In some cases, the size of the potential answer exceeds the 1000-character limit set by TREC LiveQA task. Therefore, we limit the number of characters to 1000 character at maximum. We truncate those answers by returning the first k sentences whose total size is less than 1000 character.

5. **Rank the potential answers.**

   We used a heuristic based on some of the features defined in [5]:

   - **Overall Match(AM):** Number of non-stop question terms matched in the complete answer.
   - **Answer Span(AS):** The largest distance (in words) between two non-stop question words in the answer.
   - **Same Word Sequence(WS):** The number of non-stop question words that are recognized in the same order in the answer.

[4]https://lucene.apache.org/

We score every possible answer a_i with respect to a question q as follows:

$$score(q, a_i) = AM(q, a_i) + WS(q, a_i) - AS(q, a_i) \quad (11)$$

Finally, we return only the answer a_i with the highest score as our retrieved answer.

3.2 Evaluation

3.2.1 Evaluation Measures

The evaluation for TREC LiveQA track is based on 1087 questions. These 1087 questions were judged and scored using 4-level scale:

- 4: Excellent – a significant amount of useful information, fully answers the question.
- 3: Good – partially answers the question
- 2: Fair – marginally useful information
- 1: Bad – contains no useful information for the question
- -2: the answer is unreadable (only 15 answers from all runs were judged as unreadable)

Based on the four levels labeling, the evaluation measures adopted by TREC are:

- avg-score(0-3) – average "quality" score over all queries
- succ@i+ – number of questions with i+ score (i = 1..4) divided by number of all questions
- prec@i+ – number of questions with i+ score (i = 2..4) divided by number of answered only questions

3.2.2 Results

We present our results as reported by TREC in table 4. The first column names the evaluation measure, the second column shows our reported results, and third column shows the average results over all submitted runs of the different participating teams in the track. We show in table 5 the breakdown of the labels given to our system. We compare our breakdown to the average reports over the participating systems in the track. These percentages are calculated using the reported **succ@i+** measures.

3.2.3 Discussion

The behavior of our system was fairly expected, as we did not incorporate more sophisticated steps in the body of our pipeline. Several enhancements can be applied in many parts of the system. For example, the system we presented uses only Yahoo! Answers; this is a limiting choice since there are more sources like Google web search, Wikipedia, and Quora that can be leveraged as well. Also, the ranking

Table 4: QU-Results for LiveQA task compared to the average of the submitted runs.

Evaluation Measure	QU	Track Average
avg score (0-3)	0.256	0.465
succ@1+	0.995	0.925
succ@2+	0.163	0.262
succ@3+	0.070	0.146
succ@4+	0.023	0.060
prec@2+	0.164	0.284
prec@3+	0.070	0.159
prec@4+	0.023	0.065

Table 5: Labels of QU run for LiveQA task compared to the average of the submitted runs.

Label	QU	Track Average
Excellent(4)	2.3%	6%
Good(3)	4.7%	8.6%
Fair(2)	9.3%	11.6%
Bad/Unrecognized(1& -2)	83.7%	73.8%

function is just a heuristic. We can use a more principled way to rank the answers using a learning to rank approach. Adding enhancements to the basic system we presented here might improve the performance in the future runs of the LiveQA track.

4. CONCLUSION

In this work we presented the real-time systems developed for the Microblog and LiveQA Tracks at TREC-2015. The main focus of our tweet filtering systems was to experiment with different relevance threshold modes: static and dynamic thresholds. In addition, we performed expansion of interest profiles to enrich the profile representation while giving the profile's title the largest influence in order to avoid drifting from the original topic. The expansion was applied in both push notification and email digest scenarios. The results show that the runs with static thresholds and light or no expansion in both scenarios outperform the other runs on all evaluation measures. For our future work, we plan to deeply investigate the reasons behind the relatively poor performance of scenario B by running more experiments. Additionally, we plan to experiment with re-ranking the results returned by the Lucene search engine using cosine similarity in order to maintain consistency with the relevance similarity method used in scenario A.

For Live QA track, we implemented a very simple system for the question-answering task; for a given posted question, we retrieved answers to questions that are similar to the asked question but were posted in the past; we then rank those answers using a heuristic approach and return one within the limit of 1000 characters imposed by the track. The system can be considered as a baseline for our future work with many possible directions for improvements that include enriching the sources of the answers and incorporating a more principled way for ranking answers.

5. ACKNOWLEDGMENTS

This work was made possible by NPRP grant# NPRP 6-1377-1-257 and NPRP grant# NPRP 7-1313-1-245 from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

6. REFERENCES

[1] M. Albakour, C. Macdonald, I. Ounis, et al. On sparsity and drift for effective real-time filtering in microblogs. In *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*, pages 419–428. ACM, 2013.

[2] J. Lin. Trec 2015 track guidelines. `https://github.com/lintool/twitter-tools/wiki/TREC-2015-Track-Guidelines`.

[3] J. Lin, R. Snow, and W. Morgan. Smoothing techniques for adaptive online language models: topic tracking in tweet streams. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 422–429. ACM, 2011.

[4] I. Soboroff, I. Ounis, J. Lin, and I. Soboroff. Overview of the trec-2012 microblog track. In *Proceedings of TREC*, volume 2012, 2012.

[5] M. Surdeanu, M. Ciaramita, and H. Zaragoza. Learning to Rank Answers on Large Online QA Collections. In *ACL*, pages 719–727, 2008.